Bonjour Overview

Networking, Internet, & Web: Services & Discovery

2006-05-23

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

Figures and Tables

Introduction to Bonjour Overview

Part of Mac OS X's Bonjour zero-configuration networking architecture is a powerful new system for publishing and discovering services on an IP network. This system finally brings simplicity and ease-of-use together with the power of TCP/IP network services.

This document gives a high level overview of how Bonjour works. It is not meant as a programming guide. Instead, this document covers the architecture of Bonjour and gives an overview of the different Application Layer Interfaces (APIs) that are available.

Organization of This Document

This document contains the following articles:

- "About Bonjour" (page 9) describes the problems that Bonjour solves and how it solves them.
- "Domain Naming Conventions" (page 17) explains Internet domains names, the Bonjour local domain, and the naming rules for Bonjour services instances and services types.
- "Bonjour Operations" (page 21) describes the architecture for service publication browsing.
- "Bonjour API Architecture" (page 29) describes the three network service API layers.

See Also

- *DNS Service Discovery Programming Guide* describes the Bonjour API appropriate for Darwin and Windows programmers and developers.
- *NSNetServices and CFNetServices Programming Guide* describes the Bonjour API appropriate for Cocoa programmers and C and C++ programmers on Mac OS X.

About Bonjour

Bonjour is an open protocol for zero-configuration networking over IP that Apple has submitted to the IETF as part of the ongoing standards-creation process. This section describes the problems that Bonjour solves and how it solves them.

Why Bonjour?

Over the last twenty years, a large number of wide-area networking protocols have appeared and disappeared. In recent years, Internet protocol (IP) has become the single predominant networking standard on every computing platform. The majority of computers, and many other network devices, all speak a common language. For wide-area networks and the Internet, IP protocol is all you need.

On local area networks (LANs) however, there are still some problems with using IP. Many networks have become reliant on protocols that require no system administration, such as AppleTalk. While IP has emerged as a unifying protocol for wide area networks and the Internet, it is not a universal standard on local networks, especially small networks and home networks. These networks don't often have dedicated address and name servers (DHCP and DNS) or a real system administrator.

For IP to work, every device needs a unique address. To make this happen, either everyone must agree on static IP addresses and manually type them in, or someone must set up a DHCP server or similar service to dynamically allocate addresses to clients. To refer to services by name, someone must set up a DNS server to perform the name-to-address translation, and, typically, use "well known ports" for specific types of services. To use a service, network users have to know its name, so when a service is added, everyone needs to be notified. Someone with network administration experience has to configure and maintain it.

More and more, people that do not fit the traditional role of the network administrator are setting up networks. Families are setting up home networks so they can share printers, files, and Internet connections. Peers meeting at conferences are setting up ad-hoc networks to exchange data. Even inside well-managed corporate networks, employees are adding devices to and removing devices from their local subnets. Currently, all these activities require manual configuration of IP addresses and names.

This new breed of network administrator does not want to configure subnet masks or DNS servers. Even a highly competent network administrator doesn't want to send email to every employee every time a printer is added to the network. Printer manufacturers and game publishers don't want to support multiple protocol stacks on a $50 product. People need to be able to plug in a printer, or plug two laptops together, or look for a file server or game server on the local network, without wasting time trying to get the configuration right.

Once the configuration of devices on an IP network is right, the user needs to know the exact name of any printer or other service in order to use it. That's better than typing in an IP address, but it doesn't help the user find services he or she doesn't already know about. And it doesn't forgive spelling errors. Browsing for available services is often simply impossible. A large number of IP service browsing protocols have appeared and disappeared, but none has achieved critical mass.

Before the emergence of IP as the preeminent interoperative networking protocol, AppleTalk solved the configuration and usage problems that continue to hinder IP today. With AppleTalk, users can simply browse for a service and click to choose it. For example, if you connect a group of Macintosh computers running Mac OS 9 or earlier with an Ethernet hub, they can instantly see all the available printers, file servers, and other services available on the local network. All this happens without centralized allocation of network addresses, without a centralized name server, and without a centralized repository of available services.

People need a simple and reliable way to configure and browse for services over IP networks. They want to discover available services and choose one from a list, instead of having to know each service's name or IP address in advance. It is in everyone's interest for IP to have this capability. This is exactly the capability that Bonjour provides.

Zero Configuration: An Example

Zero-configuration IP networking holds a large amount of potential. Consider the everyday task of printing. Once a printer is configured on your computer, it's simply a matter of choosing an application's Print command.

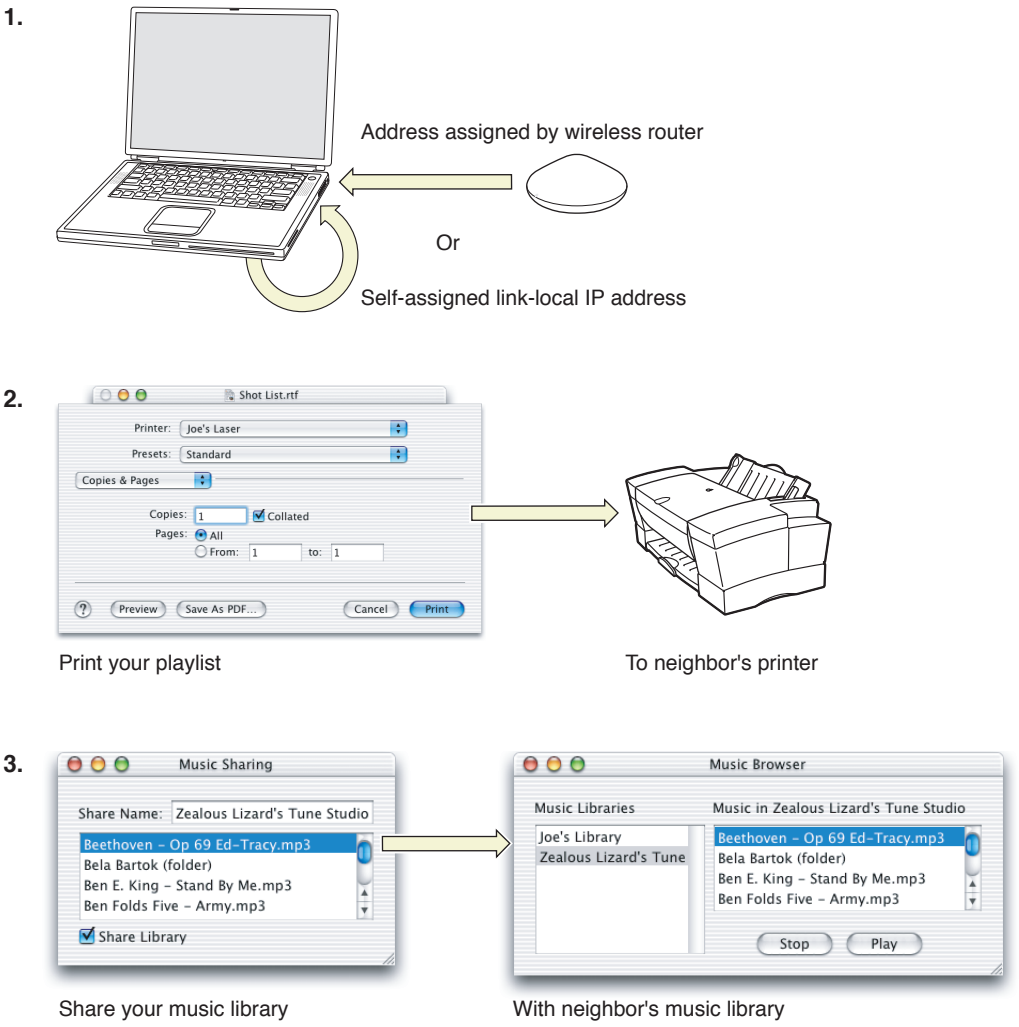
Take your laptop to a client's company, or a neighbor's house, and try to print something. If they have a printer that supports Bonjour protocols, printing is just as easy as it was on your local network. To print, connect an Ethernet cable from your laptop to your client's LAN and start up your laptop. Or start up your laptop and it instantly finds your neighbor's home wireless network. Either way, your laptop automatically discovers any available printers. You open the document, choose the Print command, and every available printer appears in the Print dialog. You select a printer, click Print, and the document prints.

Or say you want to play a network game with a friend. You open the game, and your friend's copy of the game instantly sees your copy over the network. Or if you have a music sharing application on both computers, the programs themselves can discover each other and instantly swap songlists. Similarly, if you have a shared folder or have personal Web sharing turned on, your shared files and Web pages are instantly available to others.

This scenario is illustrated in Figure 1 (page 11). In step 1, you open up your laptop in your neighbor's house, and the laptop either obtains an address from the DHCP server in the router or, in the absence of a DHCP server, assigns itself an available local address. In step 2, the network is queried for available printers so that when you open the Print dialog, your neighbor's printer is listed. Finally, in step 3, you turn on music sharing on your computer, and your neighbor's computer sees it and connects.

These are just a few of the existing applications that can benefit from zero-configuration IP networking. Zero-configuration IP networking has the potential to enhance contact management, PDA synchronization, distributed processing, and many other network applications. Additionally, zero-configuration IP networking opens the door for a whole new class of IP-enabled digital devices.

Figure 1 A typical zero-configuration networking session



What Is Bonjour?

Bonjour is Apple's proposal for zero-configuration networking over IP. Bonjour comes out of the work of the ZEROCONF Working Group, part of the Internet Engineering Task Force (IETF). The ZEROCONF Working Group's requirements and proposed solutions for zero-configuration networking over IP essentially cover three areas:

- addressing (allocating IP addresses to hosts)
- naming (using names to refer to hosts instead of IP addresses)
- service discovery (finding services on the network automatically)

Bonjour has a zero-configuration solution for all three of these areas, as described in the following four sections.

Bonjour allows service providers, hardware manufacturers, and application programmers to support a single network protocol—IP—while breaking new ground in ease of use.

Network users no longer have to assign IP addresses, assign host names, or even type in names to access services on the network. Users simply ask to see what network services are available, and choose from the list.

In many ways, this kind of browsing is even more powerful for applications than for users. Applications can automatically detect services they need or other applications they can interact with, allowing automatic connection, communication, and data exchange, without requiring user intervention.

Addressing

The addressing problem is solved by self-assigned link-local addressing. Link-local addressing uses a range of addresses reserved for the local network, typically a small LAN or a single LAN segment.

Self-assigned addressing is simply picking a random IP address in the link-local range and testing it. If the address is not use, it is now your local address. If it is already in use, pick another address and try again.

Note: Two hosts are considered to be on the same local link if, when one host sends packets to the other, the entire link-layer payload (the content of the packet as represented in the physical network, such as Ethernet) arrives unmodified. In practice, on an Ethernet network, this means that no IP router touches the packet between the two hosts.

Self-assigned link-local addressing has already shipped on IPv4 starting with Mac OS 8.5, Windows 98, and Mac OS X v10.0. The IPv6 specification includes self-assigned link-local addressing.

Any user or service on a computer that supports self-assigned link-local addressing benefits from this feature automatically. When your host computer encounters a local network, it finds an unused local address and adopts it. No action on your part is required.

Hardware manufacturers should implement self-assigned link-local addressing on their devices to obtain the full benefit of Bonjour.

Naming

The proposed solution for name-to-address translation on a local network uses Multicast DNS (mDNS), in which DNS-format queries are sent over the local network using IP multicast. Because these DNS queries are sent to a multicast address, no single DNS server with global knowledge is required to answer the queries. Each service or device can provide its own DNS capability—when it sees a query for its own name, it provides a DNS response with its own address.

Bonjour goes a bit further. It includes a responder that handles mDNS queries for any network service on the host computer. This relieves your application of the need to interpret and respond to mDNS messages. By registering your service with the Bonjour mDNSResponder daemon, any queries for your name are directed to your address automatically.

Note: Registration is performed using one of the Bonjour APIs. This is available only to services running on the host computer. Services running on other devices, such as printers, need to implement a simple mDNSResponder daemon that handles queries for services provided by that device.

For name-to-address translation to work properly, a unique name on the local network is necessary. Unlike conventional DNS host names, the local name only has significance on the local network or LAN segment. You can self-assign a local name the same way you self-assign a local address—choose one; if it's not already in use, it's yours:

- Hardware manufacturers determine whether their chosen name is already in use by having their device send an mDNS query for that name and looking for any response. If there is a response, the device should choose another name. Devices without a user interface append an incrementally larger number to a default name until the name is unique. For example, if a printer with the default name `XYZ-LaserPrinter` attaches to a local network with two other identical printers already installed, it tests for `XYZ-LaserPrinter`, then `XYZ-LaserPrinter-2`, then `XYZ-LaserPrinter-3`, which is unused and becomes its name.
- Software services provide a name when they register with Bonjour. If the provided name is already in use, Bonjour will autorename your service for you by default.

Starting with Mac OS X v10.2, users can set a host name for their computers via the Local Hostname setting in the Sharing pane of System Preferences. The host name can be used anywhere a conventional DNS host name is used—Web browsers, command line tools, and so on. To indicate to the system that a name is a local host name, append a dot (`.`) and `local.` to the host name — `Steve.local.` is an example of a local host name.

Important: The first dot acts as a separator. To prevent applications from looking for services using the search domain, fully enumerate a host name by adding the last dot in `local.`.

For example, if a user types `steve.local.` into a Web browser, this tells the system to multicast the request for `steve` on the local network instead of sending it to the conventional DNS server. If a Bonjour-enabled computer named `steve` is on the local network, the user's browser is sent the correct IP address for it. This allows users to access local hosts and services without a conventional DNS server.

Note: Users can avoid typing `.local.` after Bonjour host names by entering `local` in the Search Domains section of the Network pane in System Preferences, along with any other DNS domains such as `apple.com` or `earthlink.net`. An unqualified name, such as `steve`, is searched for in successive domains listed in the Search Domains section of the Network pane, in this case `steve.apple.com`, `steve.earthlink.net`, and `steve.local`.

For more information, see "Bonjour and Domain Names" (page 17).

Service Discovery

The final element of Bonjour is service discovery. Service discovery allows applications to find all available instances of a particular type of service and to maintain a list of named services. The application can then resolve a named instance of a service to an IP address and port number, as described in "Naming" (page 12).

The list of named services provides a layer of indirection between a service and its current IP address. Indirection allows applications keep a persistent list of available services and resolve an actual network address just prior to using a service. The list allows services to be relocated dynamically without generating a lot of network traffic announcing the change.

Service discovery in Bonjour is accomplished by "browsing." An mDNS query is sent out for a given service type and domain, and any matching services reply with their names. The result is a list of available services to choose from.

This is very different from the traditional device-centric idea of network services. For someone who deals with servers, network devices, and network programming, it is easy to get in the habit of thinking about services in terms of physical hardware. In this device-centric view, the network consists of a number of devices or hosts, each with a set of services. For example, the network might consist of a server machine and several client machines. In a device-centric browsing scheme, a client queries the server for what services it is running, gets back a list (FTP, HTTP, and so on), and decides which service to use. The interface reflects the way the physical system is organized. But this is not necessarily what the user logically wants or needs.

Users typically want to accomplish a certain task, not query a list of devices to find out what services are running. It makes far more sense for a client to ask a single question: "What print services are available?" than to query each available device with the question, "What services are you running?" and sift through the results looking for printers. The device-centric approach is not only time-consuming, it generates a tremendous amount of network traffic, most of it useless. The service-centric approach sends a single query, generating only relevant replies.

Additionally, services are not tied to specific IP addresses or even host names. For example, a website may be hosted by multiple servers with different addresses. Within an organization, network administrators may need to move a service from one server to another to help balance the load. If clients store the host name (as in most cases they now do), they will not be able to connect if the service moves to a different host.

Bonjour takes the service-oriented view. Queries are made according to the type of service needed, not the hosts providing them. Applications store service names, not addresses, so if the IP address, port number, or even host name has changed, the application can still connect. By concentrating on services rather than devices, the user's browsing experience is made more useful and trouble-free.

How Bonjour Reduces Overhead

Server-free addressing, naming, and service discovery have the potential to create a significant amount of excess network traffic, but Bonjour takes a number of steps to reduce this traffic to a minimum. This allows Bonjour to attain AppleTalk's ease of use while avoiding any unnecessary "chattiness."

Bonjour makes use of several mechanisms for reducing zero-configuration overhead, including caching, suppression of duplicate responses, exponential back-off, and service announcement, as described in the following sections.

Caching

Bonjour uses a cache of Multicast DNS records to prevent hosts from requesting information that has already been requested. For example, when one host requests, say, a list of LPR print spoolers, the list of printers comes back via multicast, so all local hosts see it. The next time a host needs a list of print spoolers, it already has the list in its cache and does not need to reissue the query. The Multicast DNS responder is responsible for maintaining the cache; application developers do not need to do anything to maintain it.

Suppression of Duplicate Responses

To prevent repeated answers to the same query, Bonjour service queries include a list of known answers. For example, if a host is browsing for printers, the first query includes no print services and gets, say, twelve replies from available print servers. The next time the host queries for print services, the query includes a list of known servers. Print servers already on the list do not respond.

Bonjour suppresses duplicate responses in another way. If a host is about to respond, and notices that another host has already responded with the same information, the host suppresses its response.

Application developers do not need to take any action to suppress duplicate responses. Bonjour handles duplicate response suppression.

Exponential Back-off and Service Announcement

When a host is browsing for services, it does not continually send queries to see if new services are available. Instead, the host issues an initial query and sends subsequent queries exponentially less often, for example: after 1 second, 2 seconds, 4 seconds, 8 seconds, and so on, up to a maximum interval of one hour.

This does not mean that it can take over an hour for a browser to see a new service. When a service starts up on the network, it announces its presence with the same exponential back-off algorithm. This way, network traffic for service announcement and discovery is kept to a minimum, but new services are seen very quickly.

Services running on a Bonjour-equipped host are announced automatically when they register with the mDNSResponder daemon. Services running on other hardware, such as printers, should implement service announcement with exponential back-off to take full advantage of Bonjour.

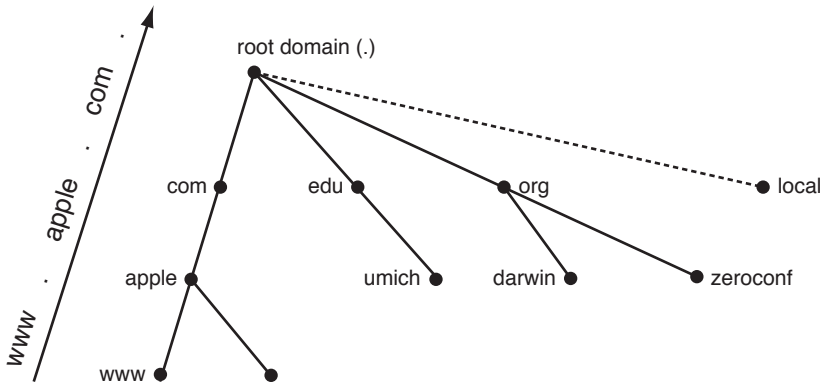
Domain Naming Conventions

Bonjour names for service instances and service types are related to Domain Name System (DNS) domain names. This section explains DNS domain names, the Bonjour local "domain," and the naming rules for Bonjour service instances and service types.

Domain Names and DNS

DNS uses a specific-to-general naming scheme for domain names. The most general domain is `.` ("dot"), called the **root domain**, which is akin to the root directory `/` in a UNIX file system. Every other domain falls in a hierarchy below the root domain. For example, the name `www.apple.com.` is within the **second-level domain** `apple.com.`, which is within the **top-level domain** `com.`, which in turn is part of `.` ("dot"), the root domain. Figure 1 shows an abridged version of this hierarchy.

Figure 1 Part of the Internet Domain Name System, augmented for Bonjour



At the top of the inverted tree is the root domain. Below it are some of the top-level domains: `com.`, `edu.`, and `org.`, and the local Bonjour "domain" `local.`, discussed further in "Bonjour and the Local Link" (page 18). Below the top level are a few second-level domains, `apple`, `darwin`, and `zeroconf`. The tree can extend infinitely downward with, for example, `www`, at the third level.

You may have noticed that the trailing dot is left off of most domain names. The trailing dot does, however, have meaning. A domain name ending in a trailing dot, such as `www.apple.com.`, is known as a **fully qualified domain name**, much like an absolute path (such as `/usr/bin`) in a UNIX file system.

If you type `wibble.apple.com` into your Web browser (without the trailing dot), the system will treat it as an unqualified (partial) name and append the names from your list of search domains, such as `apple.com.`, `earthlink.net.`, `myschool.edu.`, etc. The system will first try to append `.` ("dot," the root domain), but if the name `wibble.apple.com.` doesn't exist, it will continue down the list and try `wibble.apple.com.apple.com.`, `wibble.apple.com.earthlink.net.`, `wibble.apple.com.myschool.edu.`, and so on, which is almost certainly not what you wanted.

Bonjour and the Local Link

Bonjour protocols deal in large part with the part of the network called the **local link**. A host's local link, or **link-local network**, includes itself and all other hosts that can exchange packets without IP header data being modified. In practice, this includes all hosts not separated by a router.

On Bonjour systems, `local.` is used to indicate a name that should be looked up using an IP multicast query on the local IP network.

Note that `local.` is not really a domain. You can think of `local.` as a pseudo-domain. It differs from conventional DNS domains in a fundamental way: names within other domains are globally unique; link-local domain names are not. There is only one logical DNS entry in the world named `www.apple.com.`, and because of the way DNS works, there can be only one. Host names ending in `local.`, on the other hand, are managed by a collection of Multicast DNS responders on the local network, so the naming scope is just that: local. There can easily be two hosts named `meow.local.` in the world, or even the same building, just not on the same local network.

Globally unique names are important and useful—in fact, they are one of the significant achievements of the Internet—but they require a certain level of administrative effort to set up and maintain. Local names are useful only on the local network, but in cases where that is adequate, they provide a way to refer to network devices using names instead of IP numbers, and of course they require less effort and expense to coordinate than globally unique names.

Locally unique names are particularly useful on networks that have no connection to the global Internet, either by design or because of interruption, and on small, temporary networks, such as a pair of computers linked by a crossover cable, or a few people playing network games using laptops on the wireless network of a home or cafe.

If a name collision on the local network occurs, a Bonjour host finds a new name automatically (in the case of a device without a screen) or by asking the user (in the case of a personal computer).

Bonjour Names for Existing Service Types

Bonjour services are named according to the existing Internet standard for IP services. The website http://www.dns-sd.org keeps a registry of TCP and UDP protocol names and ports assigned to each, and Bonjour services are named according to this list. The list used by Mac OS X can be found in `/etc/services`, and the most current version of the list is available athttp://www.dns-sd.org/ServiceTypes.html.

Bonjour service names combine service types and transport protocols to form a registration type. The registration type is used to register a service and create DNS resource records for it. To distinguish registration types from domain names in DNS resource records, registration types use underscore prefixes to separate the components that make up a registration type. The format is

_ServiceType.*_TransportProtocolName*.

The service type is the official IANA-registered name for the service, for example, `ftp`, `http` or `printer`. The transport protocol name is `tcp` or `udp`, depending on the transport protocol the service uses. An FTP service running over TCP would have a registration type of `_ftp._tcp`. and would register a DNS PTR record named `_ftp._tcp.local.` with its hosts' Multicast DNS responder.

Bonjour Names for New Services

If you are designing a new protocol to advertise as a Bonjour network service, you should register it at http://www.dns-sd.org/ServiceTypes.html.

The IANA currently requires that every registered service be associated with a "well-known port" or range of well-known ports. For example, `http` is assigned port `80`, so that whenever you visit a website in your web browser, the application assumes that the HTTP service is running on port `80` unless you specify otherwise. This way, the port number for a website need only be memorized if the website is configured in a non-standard way.

With Bonjour, however, you don't have to know about port numbers. Because client applications can discover your service with a simple query for the service type, well-known ports are unnecessary.

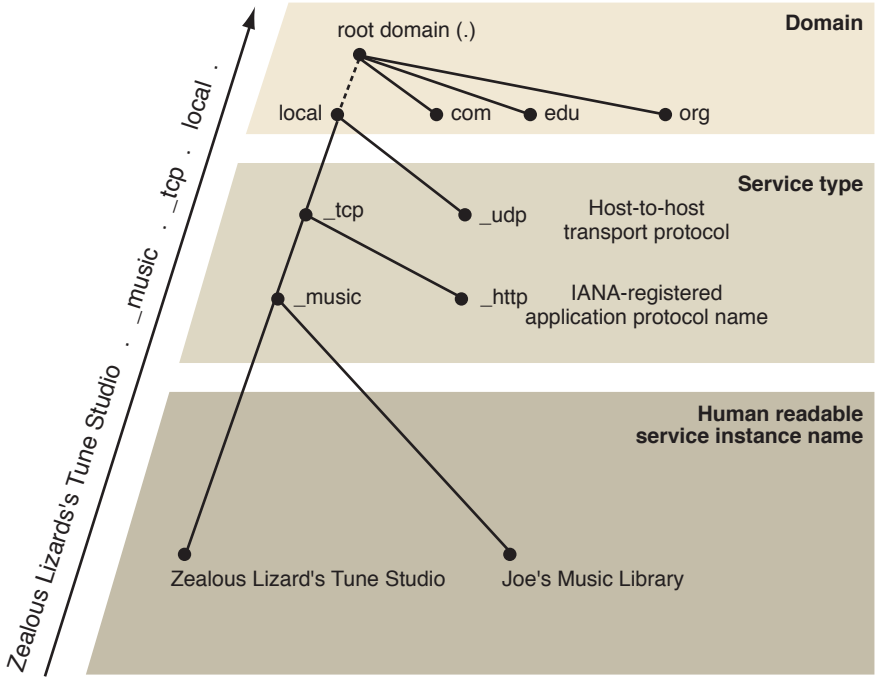
Bonjour Names for Service Instances

Service instance names are intended to be human-readable strings. As such, you should name them descriptively, and let the user override whatever default name you provide. Because they are intended to be browsed rather than typed, service instance names can be any Unicode string encoded with UTF-8, up to 63 octets (bytes) in length.

For example, an application for sharing music over the network might use the local user's name for a music sharing service, such as `Ed's Music Library`, by default. The user could override the default and name the service `Zealous Lizard's Tune Studio`, and the application would register a DNS SRV record named `Zealous Lizard's Tune Studio._music._tcp.local.`, assuming the application's music sharing protocol was associated with the name `music`.

Figure 2 (page 20) illustrates the organization of the name of a Bonjour service instance. At the top level of the tree is the domain, such as `local.` for the local network. Below the domain is the registration type, which consists of the service type preceded by an underscore (`_music`) and the transport protocol, also preceded by an underscore (`_tcp`). At the bottom of the tree is the human-readable service instance name, such as `Zealous Lizard's Tune Studio`. The complete name is a path along the tree from bottom to top, with each component separated by a dot.

Figure 2 Organization of a Bonjour service name



Bonjour Operations

This article describes the Bonjour operations of service publication, browsing, and resolution that underlay the three network service API layers, and the API layers themselves. You should read this article if you want to write an application or tool that publishes or discovers network services.

Architectural Overview

The network services architecture in Bonjour includes an easy-to-use mechanism for publishing, discovering, and using IP-based services. Bonjour supports three fundamental operations, each of which is a necessary part of zero-configuration network services:

- Publication (advertising a service)
- Discovery (browsing for available services)
- Resolution (translating service names to addresses and port numbers for use)

These operations are discussed in detail in the following sections.

Publication

To publish a service, an application or device must register the service with a Multicast DNS responder, either through a high-level API or by communicating directly with the responder (mDNSResponder in Mac OS X v10.2 and later). Using Mac OS X v10.4 also supports storing records on conventional DNS servers as well, using Dynamic DNS Update.

When a service is registered, three related DNS records are created: a service (SRV) record, a pointer (PTR) record, and a text (TXT) record. The TXT record contains additional data needed to resolve or use the service, although it is also often empty.

Service Records

The SRV record maps the name of the service instance to the information needed by a client to actually use the service. Clients then store the service name as a persistent way to access the service, and perform a DNS query for the host name and port number when it's time to connect. This additional level of indirection provides for two important features. First, the service is identified by a human-readable name instead of a domain name and port number. Second, clients can access the service even if its port number, IP address, or host name changes, as long as the service name remains the same.

The SRV record contains two pieces of information to identify a service:

- Host name
- Port number

The host name is the domain name where the service can currently be found. The reason a host name is given instead of a single IP address is that it could be a multi-homed host with more than one IP address, or it could have IPv6 addresses as well as IPv4 addresses, and so on. Identifying the host by name allows all these cases to be handled gracefully.

The port number identifies the UDP or TCP port for the service.

SRV records are named according to the following convention:

```
<Instance Name>.<Service Type>.<Domain>
```

`<Instance Name>`, the name of a service instance, can be any UTF-8-encoded Unicode string, and is intended to be human readable.

`<Service Type>` is a standard IP protocol name, preceded by an underscore, followed by the host-to-host transport protocol (TCP or UDP), also preceded by an underscore. For example, a Trivial FTP service running over UDP would have a service type of `_tftp._udp`, and an IPP printing service running over TCP would register under the `_ipp._tcp` service type. The list of official protocol names is maintained on http://www.dns-sd.org—see "Bonjour and Domain Names" (page 17) for more information.

`<Domain>` is a standard DNS domain. This may be a specific domain, such as `apple.com.`, or the generic suffix `local.` for a service accessible only on the local link.

Here is an example of the SRV record for a print spooler named `PrintsAlot` running on TCP port `515`:

```
PrintsAlot._printer._tcp.local. 120 IN SRV 0 0 515 blackhawk.local.
```

This record would be created on the Multicast DNS responder of a printer called `blackhawk.local.` on the local link. (The initial `120` represents the time-to-live—TTL—value which is used for caching.)

For more information about domain, service, and instance names, see "Bonjour and Domain Names" (page 17).

Pointer Records

PTR records enable service discovery by mapping the type of the service to a list of names of specific instances of that type of service. This record adds yet another layer of indirection so services can be found just by looking up PTR records labeled with the service type.

The record contains just one piece of information, the name of the service (which is the same as the name of the SRV record). PTR records are accordingly named just like SRV records but without the instance name:

```
<Service Type>.<Domain>
```

Here is an example of a PTR record for a print spooler named `PrintsAlot`:

```
_printer._tcp.local. 28800 PTR PrintsAlot._printer._tcp.local.
```


Publication
2006-05-23 |

Text Records

The TXT record has the same name as the corresponding SRV record, and can contain a small amount of additional information about the service instance, typically no more than 100–200 bytes at most. This record may also be empty. For example, a network game could advertise the name of the map being used in a multiplayer game, and a chat program could advertise the availability of the user (for example, idle, away, or available). If you need to transmit larger amounts of data, the host should establish a connection with the client and send the data directly.

Historically, this record has been used for multiple services running on the same port at the same IP address, for example multiple print queues running on the same print server. In this case additional information in the TXT record can be used to identify the intended print queue, as shown in this example:

Table 1 Using TXT records for demultiplexing

Service name	Type	IP address	Port	Queue name (from TXT record)
`Paper Printer`	`_ipp._tcp`	`10.0.0.2`	`631`	`rp=lpt1`
`Slide Printer`	`_ipp._tcp`	`10.0.0.2`	`631`	`rp=lpt2`

This kind of practice was necessary because service types have historically been associated with well known ports. Designers of new Bonjour protocols are encouraged to run each instance of their service on a different dynamically allocated port number, instead of trying to run them all on the same well known port number and using extra information to specify which instance the client is trying to talk to.

The nature and format of the data in the TXT record are specific to each type of service, so each new service type needs to also define the format of data for its associated TXT record (if any), and publish this format as part of the protocol specification.

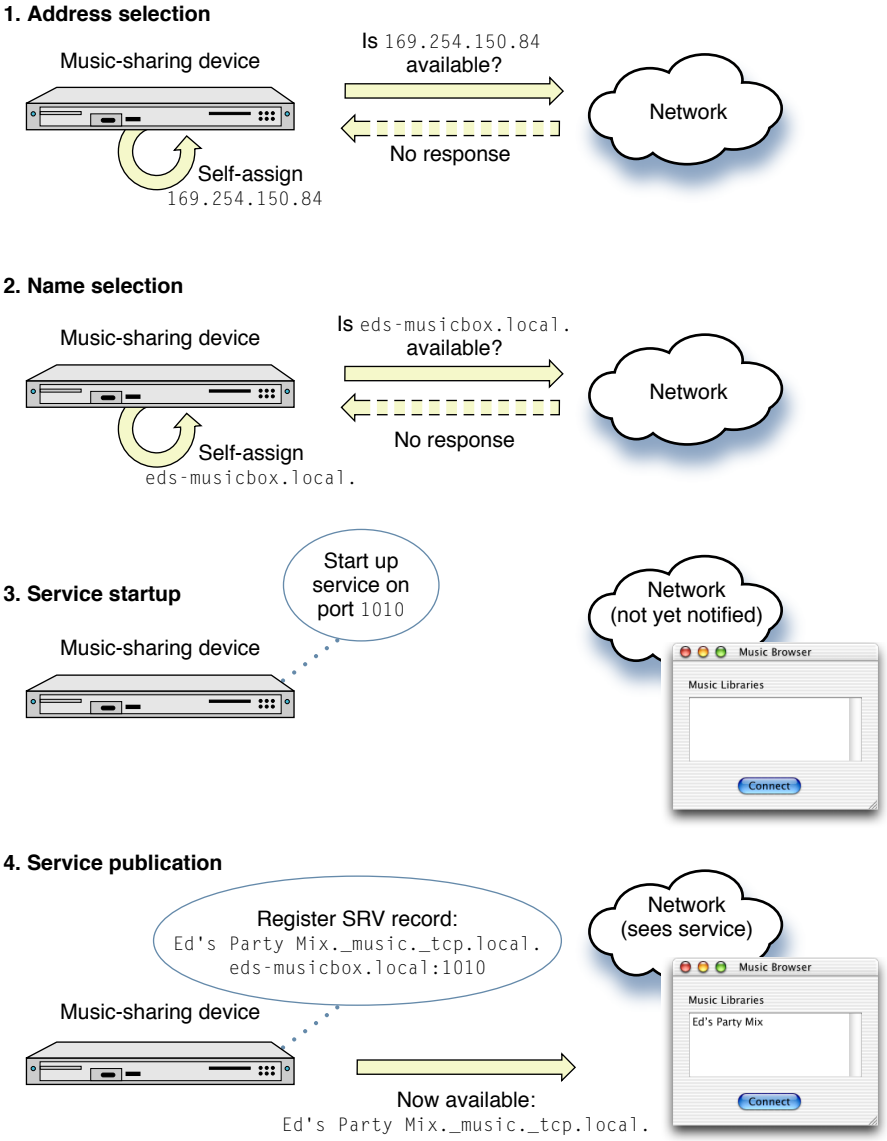
Publication: An Example

For a concrete example, consider a hypothetical device that shares music over a local network—an IP-enabled jukebox. Suppose that its transport protocol is TCP and its application protocol goes by the name `music`. When someone plugs the device into an Ethernet hub, a number of things happen, as shown in Figure 1.

In step 1, the device randomly selects the link-local IP address `169.254.150.84`, randomly selected from the IPv4 link-local range `169.254.0.0` with a subnet mask of `255.255.0.0`, and announces it to the network. Because no devices respond to the announcement, the device takes the address as its own. In step 2, it starts up its own Multicast DNS responder, requests the host name `eds-musicbox.local.`, verifies its availability, and takes the name as its own. Next, in step 3, the device starts up a music sharing service on TCP port `1010`. Finally, in step 4, it publishes the service, of type `_music._tcp`, under the name `Ed's Party Mix`, in the `local.` domain, first making sure that no service exists under the same name. This creates a SRV record named `Ed's Party Mix._music._tcp.local.` pointing to `eds-musicbox.local.` on TCP port `1010`, and a PTR record named `_music._tcp.local.`, pointing to the `Ed's Party Mix._music._tcp.local.` service.

If the requested IP address is unavailable, a host should assign itself a new one. If the domain name or service name are unavailable, a device without an interface should find a new name. Application software encountering this situation should present a user interface informing the user that the name is unavailable, and allow them to choose a different one.

Figure 1 Publishing a music sharing service



Discovery

Service discovery makes use of the DNS records registered during service publication to find all named instances of a particular type of service. To do this, an application performs a query for PTR records matching a service type, such as `_http._tcp`, usually through a higher-level API. The Multicast DNS responders running on each device return PTR records with service instance names.

Figure 2 Discovering music sharing services

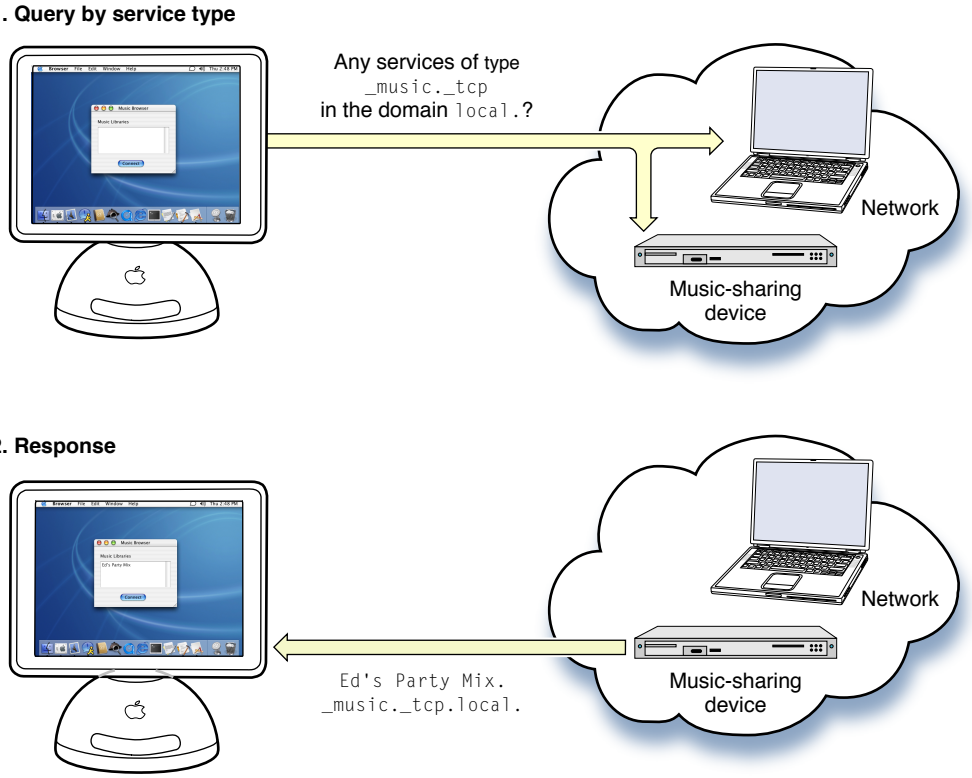


Figure 2 illustrates a client application browsing for music sharing services. In step 1, the client application issues a query for services of type `_music._tcp` in the `local.` domain to the standard multicast address `224.0.0.251`. Every Multicast DNS responder on the network hears the request, but only the music sharing device responds with a PTR record (step 2). The resulting PTR record holds the service instance name, `Ed's Party Mix._music._tcp.local.` in this case.

Resolution

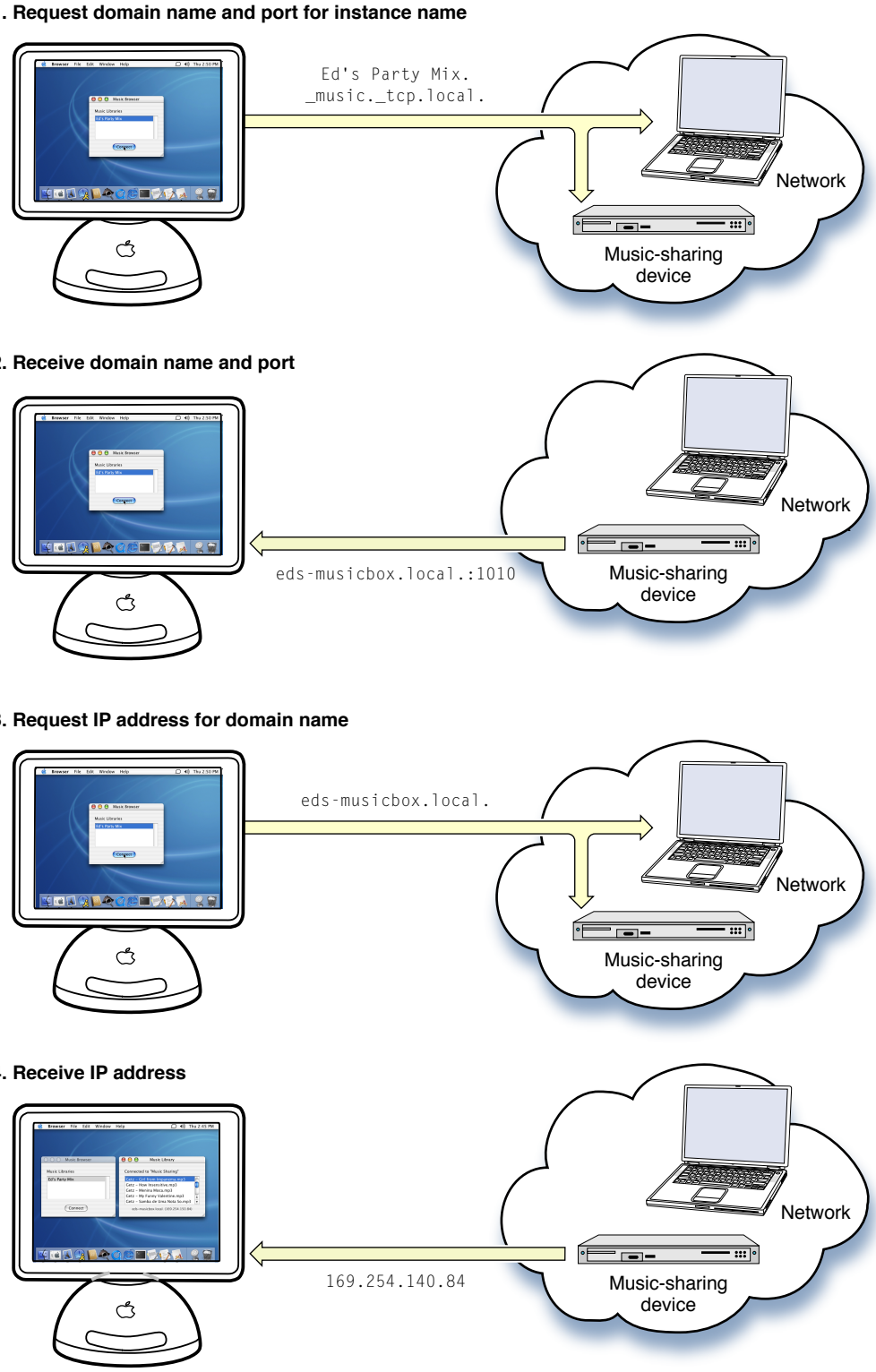
Service discovery typically takes place only once in a while—for example, when a user first selects a printer. This operation saves the service instance name, the intended stable identifier for any given instance of a service. Port numbers, IP addresses, and even host names can change from day to day, but a user should not need to reselect a printer every time this happens. Accordingly, resolution from a service name to socket information does not happen until the service is actually used.

To resolve a service, an application performs a DNS lookup for a SRV record with the name of the service. The Multicast DNS responder responds with the SRV record containing the current information.

Figure 3 illustrates service resolution in the music sharing example. The resolution process begins with a DNS query to the multicast address `224.0.0.251` asking for the `Ed's Party Mix._music._tcp.local.` SRV record (step 1). In step 2, this query returns the service's host name and port number (`eds-musicbox.local.`, `1010`). In step 3, the client sends out a multicast request for the IP address. In step 4, this request resolves to

the IP address `169.254.150.84`. Then the client can use the IP address and port number to connect to the service. This process takes place each time the service is used, thereby always finding the service's most current address and port number.

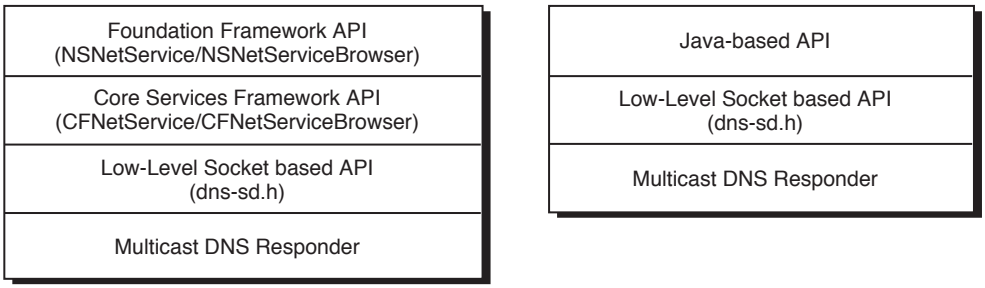
Figure 3 Resolving a music sharing service instance



Bonjour API Architecture

Mac OS X provides several layers of application programming interface (API) for Bonjour service applications: The NSNetService and NSNetServiceBrowser classes in the Foundation framework; CFNetServices, part of the CFNetwork framework in Core Services; DNS Service Discovery for Java; and the low-level DNS Service Discovery API built around BSD sockets. All three API sets provide facilities for publication, discovery, and resolution of network services. Figure 1 (page 29) illustrates the structure of the API layers. As you can see, the Multicast DNS responder (or other DNS server) sits at the lowest level, so your software does not have to interact directly with DNS.

Figure 1 API layers for Bonjour network services



NSNetService and NSNetServiceBrowser

The NSNetService and NSNetServiceBrowser classes, part of the Foundation framework in Cocoa, provide object-oriented abstractions for service discovery and publication. NSNetService objects represent instances of Bonjour services, either for publication or a service discovered by a client, and NSNetServiceBrowser represents a browser for a particular type of service. Most Cocoa programmers should find these classes sufficient to meet their needs. If you need more detailed control, you can use the DNS Service Discovery API from a Cocoa application.

NSNetService and NSNetServiceBrowser are scheduled on the default NSRunLoop object to perform publication, discovery, and resolution asynchronously. All results returned by NSNetService and NSNetServiceBrowser objects are handled by delegate objects. These objects must be associated with a run loop to function, but it need not be the default one.

CFNetServices

The CFNetServices API declared in the Core Services framework provide Core Foundation-style types and functions for managing services and service discovery. CFNetServices defines three Core Foundation object types, CFNetService, CFNetServiceBrowser, and CFNetServiceMonitor. CFNetService is an abstract representation

of a service instance, either for publication or for use. Associated functions provide support for publishing and resolving services. CFNetServiceBrowser represents a browser for a particular type of service in a particular domain. You should use this API from C or C++ applications for Mac OS X.

Both CFNetService and CFNetServiceBrowser objects are normally serviced in CFRunLoops. To retrieve results, applications implement callback functions to handle events, such as new services appearing or disappearing, instances being resolved, and errors occurring. Unlike NSNetService and NSNetServiceBrowser, CFNetServices types do not require a run loop and can run synchronously if this behavior is needed. However, it is bad practice to use the synchronous modes of these functions.

DNS Service Discovery

The DNS Service Discovery API, declared in `</usr/include/dns_sd.h>`, provide low-level BSD socket communication for Bonjour services. DNS Service Discovery acts as an intermediate layer between your software and the Multicast DNS responder or DNS server. It manages the Multicast DNS responder for you, letting you write your program in terms of services and service browsers instead of DNS resource records.

Because the DNS Service Discovery API is part of the Darwin open source project, you should use it when programming for Darwin. In some cases, using this API is simpler than CFNetServices, but DNS Service Discovery does not provide the same integration with the Core Foundation programming model.

DNS Service Discovery is also the API that should be used if developing Bonjour service applications for Windows, Linux, or FreeBSD.

Document Revision History

This table describes the changes to *Bonjour Overview*.

Date	Notes
2006-05-23	Updated Related Documents section.
2005-11-09	Removed Preliminary stamp.
2005-10-04	Changed to be an overview of the Bonjour technology.
2005-04-29	Changed "Rendezvous" to "Bonjour."
2004-08-31	Clarified use of empty string when resolving services and added details on how to correctly process the address-resolution delegate method of the NSNetService class.
	Corrected example in *Resolving and Using Network Services* to not use empty string to refer to the local domain.
	Added details on how to correctly process the `netServiceDidResolveAddress:` delegate method of NSNetService.
2004-03-09	Clarified in "Bonjour Network Services Architecture" (page 21) that a TXT record is always created, but it may be empty.
2004-03-08	Reorganized topic introduction. Added pointer to *DNS Service Discovery API*.
2002-11-12	Revision history was added to existing document.